Modelling Zoned RAID Systems using Fork-Join Queueing Simulation

Abigail S. Lebrecht[⋆], Nicholas J. Dingle, and William J. Knottenbelt

Department of Computing, Imperial College London,
South Kensington Campus, SW7 2AZ, United Kingdom
{asl102,njd200,wjk}@doc.ic.ac.uk

Abstract. RAID systems are ubiquitously deployed in storage environments, both as standalone storage solutions and as fundamental components of virtualised storage platforms. Accurate models of their performance are crucial to delivering storage infrastructures that meet given quality of service requirements. To this end, this paper presents a flexible fork-join queueing simulation model of RAID systems that are comprised of zoned disk drives and which operate under RAID levels 01 or 5. The simulator takes as input I/O workloads that are heterogeneous in terms of request size and that exhibit burstiness, and its primary output metric is I/O request response time distribution. We also study the effects of heavy workload, taking into account the request-reordering optimisations employed by modern disk drives. All simulation results are validated against device measurements.

1 Introduction

RAID[1] has revolutionised data storage because of its ability to synthesise a set of low-cost commodity storage devices into a single logical unit that can deliver high reliability with high performance. However, RAID system performance varies heavily in practice, depending on chosen configuration and operating context. Given a budget and an expected workload, it is therefore a major challenge for system designers and engineers to select RAID components and corresponding configurations capable of delivering a required level of quality of service. Performance models provide a low-cost means to evaluate the suitability of candidate system designs ahead of implementation.

In the above context, this paper introduces a queueing-based simulator for the analysis of RAID systems comprised of zoned disks. Our goal is to provide an elegant high-level framework that avoids very detailed low-level device simulation (e.g. as performed by the DiskSim [2] and RaidSim [3] simulators) and which can be simply parameterised from disk drive technical specifications.

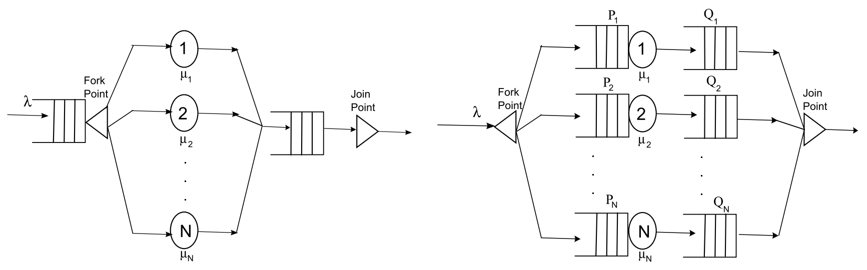
[⋆] Corresponding author. Telephone: +44 20 7594 8251.

[1] Redundant Array of Inexpensive Disks [1]; RAID levels describe various ways of spreading data across multiple storage devices using striping, mirroring, and/or parity

The simulation generates as its primary output metric the cumulative distribution function of I/O request response time. From this, it is straightforward to calculate metrics typically encountered in Service Level Agreements, including response time quantiles and the mean, variance and higher moments of I/O request response time.

Simulation is often used to study RAID system performance, since there exist no exact analytical models of RAID of any level [4]. However, there are numerous analytical queueing network approximations (e.g. [5–9]). In [4, 10, 11] we have developed approximate analytical queueing models of RAID 01 and 5. Simulations are often used to validate the results of analytical models. Additionally, they provide the ability to replicate the details of the scheduling algorithms and mechanical behaviour of real systems, while analytical models must abstract these details. Thus, simulations can aid the development of more realistic analytical models.

In [12] we introduced a zoned RAID simulator for RAID level 0 (striping, no redundancy). This simulation is based on modelling each disk drive as an $M/G/1$ queue and approximates RAID 0 as a split-merge queueing system (see Figure 1(a)). In this system, a job (I/O request) splits into N subtasks which are serviced in parallel. Only when all the subtasks finish servicing and rejoin can the next job split into subtasks and start servicing. However, it is generally accepted that the queueing model which most accurately reflects the behaviour of RAID systems is the fork-join queueing network [5]. In a fork-join queue with N queues, (see Figure 1(b)), each incoming job is split into N subtasks at the fork point. Each of these subtasks queues for service at a parallel service node before joining a queue for the join point. When all N subtasks in the job are at the head of their respective join queues, they rejoin (synchronise) at the join point.



(a) Split-merge queueing model (b) Fork-join queueing model

Fig. 1. Split-merge vs. fork-join queueing models

This paper presents a fork-join simulation capable of modelling RAID levels 01 (mirror of stripes) and 5 (distributed parity). In order to simulate a RAID system, we must first implement an effective single disk simulation. We can then use several instances of this single disk simulator as components in our disk array simulator. Section 2 summarises our single disk simulation, which utilises the *JINQS* Java-based queueing simulation library [13]. Section 3 details the fork-join extensions to the single disk simulator required to create an effective model for RAID. This involves firstly simulating a fork-join queueing network and then tailoring it to model the specific demands of RAID 01 and 5. Furthermore we enhance the model to accept various types of workload. We then validate the accuracy of this new simulator by comparing results from several different types of workload to device measurements taken on a real RAID system.

2 Single Disk Simulation

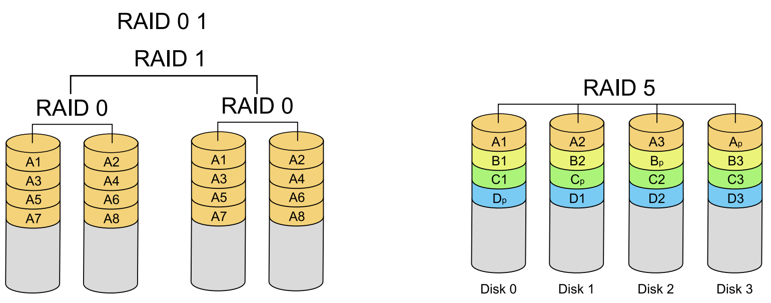
We model single disk drives as $M/G/1$ queues and use the *JINQS* Java-based simulation library [13] for $M/G/1$ queue simulation. The service time density of an access to a random location on a single disk drive is the convolution of the seek time, rotational latency and data transfer time probability density functions. In our model we use the seek time and rotational latency probability distributions defined in [14] and the data transfer time distribution from [4]. The seek, rotation and transfer times are sampled using the cumulative distribution function inversion method described in [12]. An important subtlety that needs to be taken into account is that modern disks are *zoned*, with more sectors on the outer tracks than inner tracks. Therefore, a random request is more likely to be directed to a sector on an outer track. Similarly, zoning means that it is faster to transfer data on a track close to the circumference than the centre of the disk. The seek time and data transfer models must take these factors into account. We assume that all requests are random accesses and therefore it is always necessary to position the disk head before transferring data.

3 RAID Simulation

Disk arrays organise multiple independent disks into a single logical disk unit. By striping data across multiple disks and accessing the disks in parallel, higher data transfer rates are achieved, especially with larger I/O requests. Data striping also ensures that data is balanced across the disks, avoiding data hot spots. Disk striping involves writing data blocks of a constant pre-defined size to successive disks in a cyclical pattern.

However, the larger the disk array, the more likely it is that a member disk will fail. In order to avoid data loss as a result of failures, redundancy can be employed using mirroring (see Figure 2(a)) or parity blocks (see Figure 2(b)). Parity is block-interleaved and distributed across all disks.

All these schemes involve striping of I/O accesses across disks in the disk array. A fork-join queue in which customers represent I/O requests provides a good foundation for an abstraction of this behaviour.



(a) RAID 01 (b) RAID 5

Fig. 2. RAID Configurations [15]

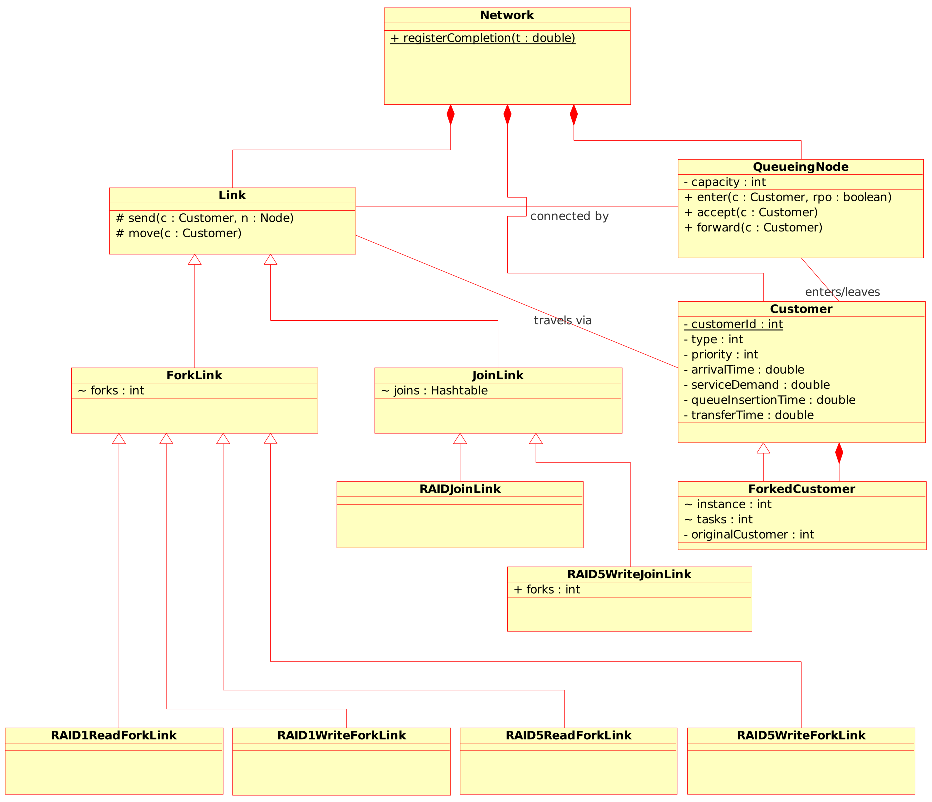


Fig. 3. RAID simulator class diagram

3.1 Fork-Join Simulation

As shown in the UML class diagram of Figure 3, our queueing simulation is specified in terms of *QueueingNode*, *Link* and *Customer* classes. *QueueingNode*s are connected by *Link*s to create a network of queues. Response times measurements are obtained by recording the time each *Customer* spends in the network. To extend this for fork-join queueing, we introduce *ForkLink* and *JoinLink* classes to extend the *Link* class and a *ForkedCustomer* class to extend the *Customer* class. *ForkedCustomer*s are created at a *ForkLink* when a customer forks into subtasks and removed at the *JoinLink*. A *ForkLink* creates a new *ForkedCustomer* for each subtask, each with a reference to the original *Customer*. These *ForkedCustomer*s are sent to one of the n single $M/G/1$ queues. When a *ForkedCustomer* leaves an $M/G/1$ queue, it is sent to *JoinLink*, which collects all *ForkedCustomer*s. When all the *ForkedCustomer*s for a particular *Customer* have arrived, the original *Customer* is sent on its way and all of its *ForkedCustomer*s are destroyed.

3.2 RAID 01 Simulation

There are certain extensions to the fork-join simulation described above that must be made to model a RAID 01 system accurately.

In particular, both the fork-join simulation described above and the RAID 0 simulation of [12] are limited to supporting requests consisting of a number of subtasks that is a multiple of the number of disks. We therefore extend the *ForkLink* class with *RAID01ReadForkLink* and *RAID01WriteForkLink* classes, both of which support striping of variable size subtasks across disks starting from a randomly selected disk. Additionally, we extend the *JoinLink* class with the *RAIDJoinLink* class to support joining of variable-sized requests.

In terms of subtask scheduling for RAID 01 read operations, we assume an efficient RAID controller which reads half the data from the primary disks and half the data from the mirror disks [7]. RAID 01 write operations send each subtask to both the primary and mirror disks and create double the number of *ForkedCustomer*s as for a read request of the same size.

3.3 RAID 5 Simulation

Manufacturers of RAID controllers seldom reveal the mechanisms and scheduling strategies involved in their products. In the cases of RAID 0 and 01, the likely disk accesses are relatively straightforward to predict. However in RAID 5, particularly with operations involving pre-reads and parity updates, there are many possibilities for scheduling strategies and disk head positioning times within a request. Here we base the design of our RAID 5 simulation upon the operational assumptions of RAID 5 disk behaviour presented in [4, 7, 11].

In a manner analogous to the RAID 01 case, we extend the *ForkLink* class with *RAID5ReadForkLink* and *RAID5WriteForkLink* classes.

A RAID 5 read request will read only from the disks containing data blocks in a stripe and not the disk with the single parity block in each stripe. To simulate this, when forking each request, the position of the parity disk is randomly chosen as well as the starting disk. If a request accesses more than one stripe, then the position of the parity disk within the array is incremented (modulo the number of disks) at the end of each stripe.

The behaviour of a RAID 5 write is complex, with different parity-update schemes that depend on the size of the request. For simplicity, we assume requests are aligned to start striping from the first disk in the array.

Given a b-block write request on an n-disk RAID 5 system, the possibilities are:

If a request consists of a number of complete stripes (i.e. $b \bmod (n-1) = 0$), all the disks are utilised, with either the new data block or the new parity block written to each disk. Full stripe writes can be simulated by sending *ForkedCustomer*s to each disk and joining them at the *RAID5WriteJoinLink* when all subtasks have completed.

If a request consists of $b \bmod (n-1) < \frac{n-1}{2}$ blocks (i.e. it consists of zero or more full stripe writes followed by a small partial stripe write), then parity is calculated using [1]:

$$\mathit{new\_parity} = \mathit{new\_data} \oplus \mathit{old\_data} \oplus \mathit{old\_parity}$$

where $\oplus$ is the exclusive-or (XOR) operator. This is a read-modify-write operation. After transferring the full stripes, each of the $b \bmod (n-1)$ blocks and parity must be transferred twice, first to read the old data and parity, then to write the new data and parity. When the old data and parity have been read from all disks, a new request will be issued to write the new data and parity to the same disks. This request is given non-preemptive priority in the queue, so at least one disk (the last to complete the pre-read) will just have completed reading a data or parity block that now needs to be re-written.

If $\frac{n-1}{2} \leq b \bmod (n-1) < n-1$ (i.e. the request consists of zero or more full stripe writes followed by a large partial stripe write), then to minimise disk accesses the parity is calculated by pre-reading from the disks that are not being written to. The new parity is calculated by XOR-ing the data that will be written with the data from the disks that will remain unchanged. This is a reconstruct-write operation. After the full stripe transfers, $n-1-b \bmod (n-1)$ blocks of data are pre-read for the calculation of the new parity. When all $n-1-b \bmod (n-1)$ disks complete their pre-read, a new request is sent to the other $b \bmod (n-1)+1$ disks to write the new data and parity.

Simulation of the above operations is supported in the *RAID5WriteForkLink* and *RAID5WriteJoinLink* classes. The *RAID5WriteForkLink* subdivides any arriving request into full stripe subtasks followed by pre-read subtasks. These subtasks are then routed to the relevant $M/G/1$ queues. When the pre-read subtasks have completed and are accounted for at the *RAID5WriteJoinLink* then, instead of completing the request, the *RAID5WriteJoinLink* creates a new high priority request to send back to the *RAID5WriteForkLink*, where it splits into $b \bmod (n-1)+1$ subtasks (the number of blocks to write plus the parity). In order for the simulation to differentiate between full stripe writes and pre-reads and the following partial stripe write, the *ForkedCustomer*s are assigned classes representing the type of request.

The subtasks of the partial stripe write will have different service times depending on the nature of the previous request serviced by the disk. In the case that $b \bmod (n-1) < \frac{n-1}{2}$, there are four possible scenarios to be considered.

The first scenario is when the disk is busy at the arrival instant of any of the partial stripe write subtasks. Since the partial stripe write is accessing all the disks used for the pre-read, and all the pre-reads must complete before the partial stripe write is issued, it is not possible that the job currently servicing is a *ForkedCustomer* from the same *Customer*. Hence to simulate a return to the required disk position to transfer data, a random sample of seek and rotation time is taken.

If the disk is idle on arrival of a subtask, then there are a further three mutually exclusive scenarios with different positioning times:

- If another request has been in service between the pre-read and partial stripe write subtasks then the simulator needs to sample a new seek and rotation time.
- If the disk was the last to complete the pre-read, then it will be positioned on the correct track, but just past the rotational position. In this case, the simulator returns a positioning time of one full disk rotation.
- Otherwise, the disk is still positioned at the correct track and the simulator needs to sample from the rotational latency for positioning time.

If $b \bmod (n-1) \geq \frac{n-1}{2}$, there are again a number of scenarios to consider. Since the pre-read involves different disks than the partial stripe write, it is possible that previous full stripe subtasks from the same request could still be servicing on the disks required for the partial stripe write after the pre-read has completed.

In this context, if a subtask arrives to a busy disk, we consider whether the job currently in service is part of the same request. If it is, the subtask will follow on with no positioning time. If it arrives to an idle disk, the simulator checks if the previous job was part of the same request. If it was then the disk head is pointing to the correct track and the simulator needs to sample rotational latency only. In all other cases the positioning time is obtained by sampling both seek and rotation time.

Since we are simulating zoned disks, we must take into account that the transfer time must be same both for the full-stripe and pre-read and for the partial stripe write requests, since they are both accessing the same position on the disk. Therefore, the transfer time for each subtask to each disk is recorded in a hash table and referred to when the partial stripe write is serviced.

When all the partial stripe write subtasks complete, the *RAID5WriteJoinLink* sends the single request on its way and removes all *ForkedCustomer*s attached to that request.

3.4 Bulk Arrivals

Most queueing simulations assume that arriving requests are Markovian. However, over the last decade, there have been many studies of storage system I/O traces (e.g. [16–21]) which consistently show that real-life arrivals to storage systems exhibit burstiness and a variety of request size distributions. Consequently, we have extended the simulator to support bulk arrivals of I/O requests at the RAID controller, making use of *JINQS*'s in-built support for arrivals that consist of a number of requests defined by a chosen probability distribution.

3.5 Rotational Positioning Optimisation

Bursty workloads [20] result in highly variable queue lengths. As queue length increases, response time suffers. To lessen this effect, many disk drives employ scheduling algorithms to reorder jobs in the queue to minimise head positioning time [22, 23]. This reduces the time needed to service each job, which in turn reduces the waiting time for all jobs [24].

We incorporate this factor into our simulation by parameterising the service time distribution sampler according to the current queue length. The sampler then takes as many combined samples of seek and rotation time as there are jobs in the queue and chooses the minimum of these to be the positioning time of the request starting service. This can be used for either single disk simulation or RAID simulation.

4 Validation

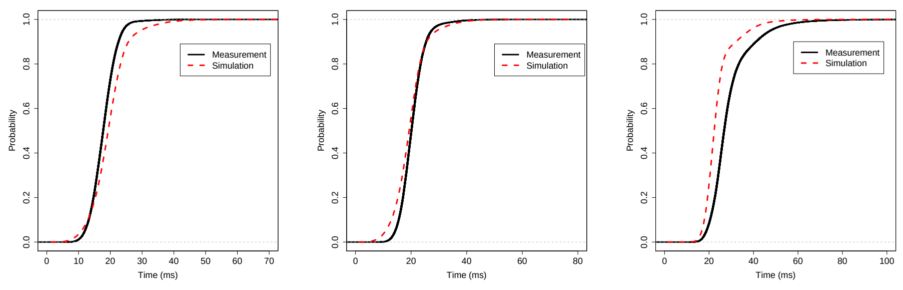
Our experimental platform consists of an Infortrend A16F-G2430 RAID system containing four Seagate ST3500630NS disks. Each disk has 60 801 cylinders. A sector is 512 bytes and we have approximated, based on measurements from the disk drive, that the time to write a single physical sector on the innermost and outermost tracks are 0.012064ms (t_{max}) and 0.005976ms (t_{min}) respectively. The stripe width on the array is configured as 128KB, which we define as the block size. Therefore there are 256 sectors per block. The time for a full disk revolution is 8.33ms. A track to track seek takes 0.8ms and a full-stroke seek requires 17ms for a read; the same measurements are 1ms and 18ms respectively for a write [25].

To obtain response time measurements from this system, we implemented a benchmarking program that issues read and write requests using a master process and multiple child processes. These child processes are responsible for issuing and timing I/O requests, leaving the master free to spawn further child processes without the need for it to wait for previously-issued operations to complete.

In order to validate the simulation model effectively, it was necessary to minimise the effects of buffering and caching as these are not currently represented in the model. We therefore disabled the RAID system's write-back cache, set the read-ahead buffer to 0 and opened the device with the `O_DIRECT` flag set. For each of the experiments presented below (both measurement and simulation), 100 000 requests were issued. We present a selection of comparisons of cumulative distribution functions (cdf). The single disk simulation is validated in [12], so here we only present RAID validations.

4.1 RAID 01

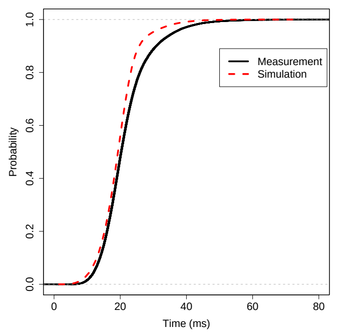
In Figure 4 we compare measurement and simulation cdfs for RAID 01 with Markovian arrivals at a rate of 0.01 requests/ms for different request type and size. We generally observe good agreement between model and measurement, particularly in Figure 4(b), in which a full stripe read is taking place. Figure 5 considers the same conditions, except in this case the request size is variable and sampled from a geometric distribution with a specified mean request size. We observe excellent agreement between model and measurement in these cases.



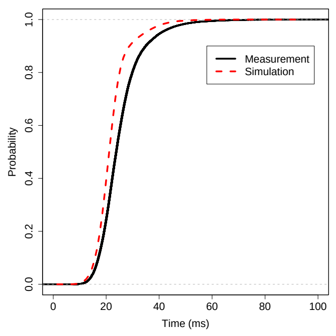
(a) 2-block read request (b) 4-block read request (c) 3-block write request

Fig. 4. Cumulative distribution functions of RAID 01 I/O request time on a 4 disk RAID system ($\lambda = 0.01$ requests/ms)

Figures 6 and 7 validate RAID 01 with more interesting workloads. In Figure 6 simulation and measurement cdfs are compared for a RAID 01 system with constant-size full stripe requests, which arrive in bursts. Each request arrives as part of a batch. The number of requests in each batch is decided by a geometric distribution. We continue to see excellent agreement between model and measurement. Figure 7 involves a high arrival rate at the array (0.06 requests/ms), such that rotational positioning optimisation (RPO) should be expected. We plot two simulation cdfs, one with RPO enabled on the simulator and the second with RPO disabled. It is clear from the graph that for large arrival rates (and hence long queue lengths) incorporating RPO into any model is crucial.



(a) 4-block mean read request



(b) 2-block mean write request

Fig. 5. Cumulative distribution functions of RAID 01 I/O request time on a 4 disk RAID system with request sizes chosen from a geometric distribution ($\lambda = 0.01$ requests/ms)

4.2 RAID 5

In Figure 8 we compare measurement and simulation cdfs for RAID 5 systems wiqth Markovian arrivals with arrival rate λ requests/ms and constant size requests. We can use these validations to help judge the accuracy of our RAID 5 models. We observe in Figure 8(a) that the read simulation appears to agree well with the measurements. Figures 8(b) and 8(e) are small partial stripe writes. While mean values appear to agree well, the shapes of the cdf curves for the simulation differ somewhat from the measurement curves, particularly in the case of a small partial stripe write that does not follow a full stripe write (Figure 8(b)). Figures 8(c) and 8(f) are large partial stripe writes. These show better agreement than the small partial stripe equivalents. However, they appear to consistently underestimate the measurements, as does the full stripe write request in Figure 8(d). It is possible that this underestimation can be attributed to not factoring into the simulation RAID controller overheads including parity computation time.

Similarly, Figure 9 compares simulation and measurements for RAID 5 requests with size decided by a geometric distribution. We again observe excellent agreement for read requests in Figure 9(a). Write requests in Figure 9(b) tend to underestimate the measurements since both full stripe requests and large partial stripe write requests of constant size underestimate the measurement. Figure 10 compares mean response times for simulation and measurement for up to 10-block jobs. The model predicts effectively the qualitative characteristics of mean RAID 5 response times as block size varies.

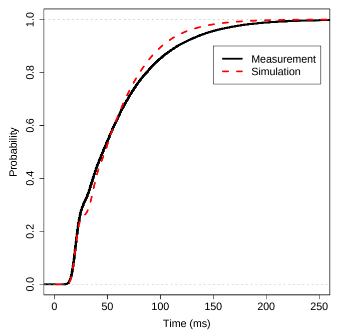


Fig. 6. Cumulative distribution functions of RAID 01 I/O read request time on a 4 disk RAID system with 4-block requests and geometrically distributed bulk arrivals with mean size 3 ($\lambda = 0.01$ requests/ms)

5 Conclusion

This paper has presented a RAID simulation based on fork-join queueing networks. We have presented extensive validations of this simulation against device measurements, generally observing excellent agreement for RAID 01 and 5 with request streams of both constant and variable size and bursty arrivals. We have also incorporated rotational positioning optimisations into our simulation and have shown in our validations that this is fundamental to any accurate representation of disk drives or RAID systems operating under heavy load.

In future work, we hope to further relax the constraints on the simulation. In particular, it is straightforward to modify the simulator to represent other RAID levels and to accept arrival streams that consist of both read and write requests, and both random and sequential accesses. Furthermore, we intend to discover more about RAID controller overheads and incorporate these into our models. We also intend to apply our simulator in validating and improving current and future analytical queueing models of RAID systems. Finally, caching is an interesting and practically useful aspect that merits further investigation and integration into our model.

References

1. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (RAID). In: Proc. International Conference on Management of Data (SIGMOD). (1988)
2. Bucy, J.S., Ganger, G.R., Contributors: The DiskSim Simulation Environment Version 3.0 Reference Manual. School of Computer Science, Carnegie Mellon University. 3.0 edn. (January 2003)

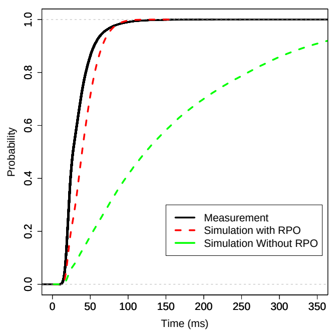
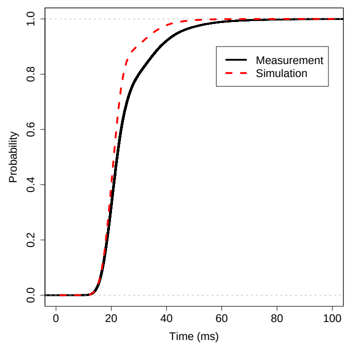
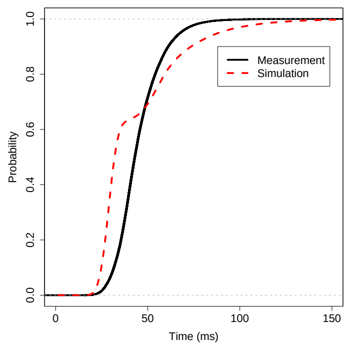


Fig. 7. Cumulative distribution functions of RAID 01 I/O read request time on a 4 disk RAID system with 4-block requests ($\lambda = 0.06$ requests/ms)

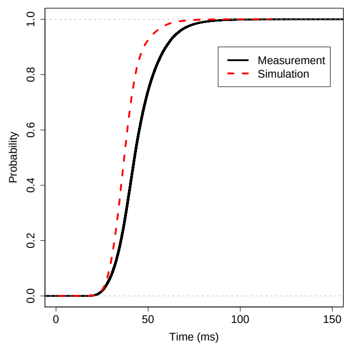
3. Chen, P.M., Lee, E.K.: Striping in a RAID level 5 disk array. SIGMETRICS Performance Evaluation Review **23**(1) (1995) 136–145
4. Lebrecht, A.S., Dingle, N.J., Knottenbelt, W.J.: A response time distribution model for zoned RAID. In: 15th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA). (June 2008)
5. Lee, E.K.: Performance Modeling and Analysis of Disk Arrays. PhD thesis, University of California at Berkeley (1993)
6. Chen, S., Towsley, D.: A performance evaluation of RAID architectures. IEEE Transactions on Computers **45**(10) (1996) 1116–1130
7. Harrison, P.G., Zertal, S.: Queueing models of RAID systems with maxima of waiting times. Performance Evaluation **64**(7-8) (August 2007) 664–689
8. Varki, E.: Response time analysis of parallel computer and storage systems. IEEE Transactions on Parallel and Distributed Systems **12**(11) (November 2001) 1146–1161
9. Varki, E., Merchant, A., Xu, J., Qiu, X.: Issues and challenges in the performance analysis of real disk arrays. IEEE Transactions on Parallel and Distributed Systems **15**(6) (June 2004) 559–574
10. Lebrecht, A.S., Dingle, N.J., Knottenbelt, W.J.: Modelling and validation of response times in zoned RAID. In: 16th IEEE International Symposium on Modeling, Analysis, and Simulationof Computer and Telecommunication Systems (MASCOTS). (September 2008)
11. Lebrecht, A.S., Dingle, N.J., Knottenbelt, W.J.: Validation of large zoned RAID systems. In: 24th UK Performance Engineering Workshop (UKPEW). (July 2008) 246–261
12. Wan, F., Dingle, N.J., Knottenbelt, W.J., Lebrecht, A.S.: Simulation and modelling of RAID 0 system performance. In: 22nd Annual European Simulation and Modelling Conference (ESM). (September 2008) 145–149
13. Field, A.J.: JINQS: An Extensible Library for Simulating Multiclass Queueing Networks. Imperial College London. (August 2006)



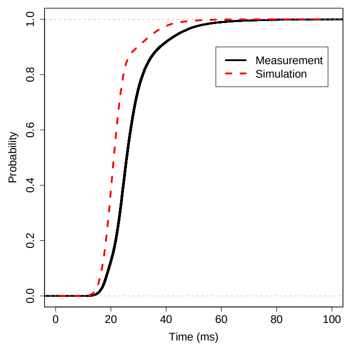
(a) 5-block read request $\lambda = 0.02$



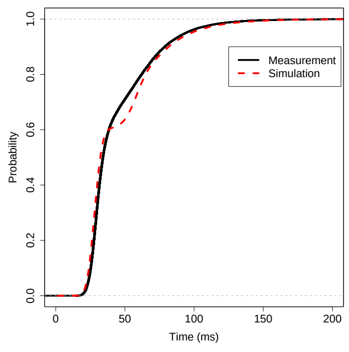
(b) 1-block write request $\lambda = 0.01$



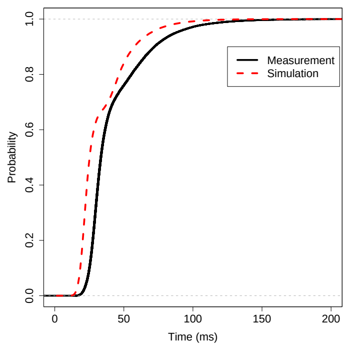
(c) 2-block write request $\lambda = 0.01$



(d) 3-block write request $\lambda = 0.01$

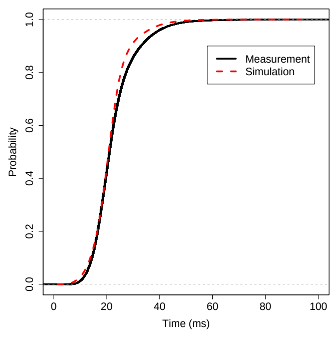


(e) 4-block write request $\lambda = 0.01$

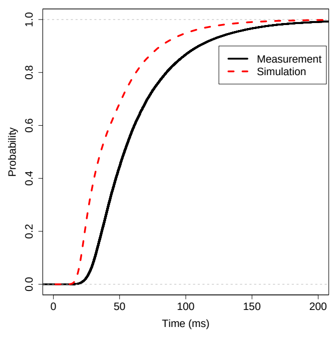


(f) 5-block write request $\lambda = 0.01$

Fig. 8. Cumulative distribution functions of RAID 5 I/O request time on a 4 disk RAID system (λ requests/ms)



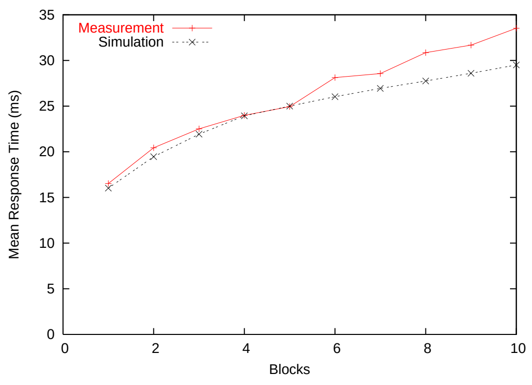
(a) 5-block mean read request, $\lambda = 0.01$



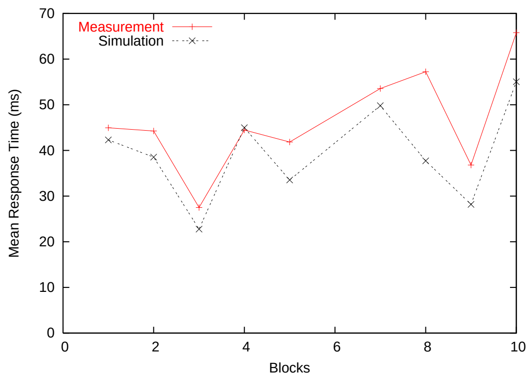
(b) 4-block mean write request, $\lambda = 0.02$

Fig. 9. Cumulative distribution functions of RAID 5 I/O request time on a 4 disk RAID system with request sizes chosen from a geometric distribution (λ requests/ms)

14. Zertal, S., Harrison, P.G.: Multi-RAID queueing model with zoned disks. In: High Performance Computing and Simulation Conference (HPCS). (June 2007)
15. Wikipedia: Standard RAID levels (April 2009) `http://en.wikipedia.org/wiki/Standard_RAID_levels`.
16. Gomez, M., Santonja, V.: Analysis of self-similarity in I/O workload using structural modeling. In: Proc. 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). (October 1999) 234–243
17. Gomez, M., Santonja, V.: Characterizing temporal locality in I/O workload. In: Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS). (2002)
18. Wang, M., Ailamaki, A., Faloutsos, C.: Capturing the spatio-temporal behavior of real traffic data. Performance Evaluation **49**(1-4) (2002) 147–163
19. Riska, A., Riedel, E.: Disk drive level workload characterization. In: Proc. USENIX '06 Annual Technical Conference (ATEC), Boston, MA (2006)
20. Ruemmler, C., Wilkes, J.: Unix disk access patterns. In: Proc. Usenix Winter Conference, San Diego, CA (1993) 405–420
21. Shriver, E., Merchant, A., Wilkes, J.: An analytic behavior model for disk drives with readahead caches and request reordering. In: Proc. ACM SIGMETRICS. (1998) 182–191
22. Jacobson, D.M., Wilkes, J.: Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7rev1, HP Laboratories (1991)
23. Seltzer, M., Chen, P., Ousterhout, J.: Disk Scheduling Revisited. In: Proc. USENIX Winter Technical Conference, USENIX Association (1990) 313–324
24. Hsu, W.W., Smith, A.J.: The performance impact of I/O optimizations and disk improvements. IBM Journal of Research and Development **48**(2) (2004) 255–289
25. Seagate: Barracuda ES Data Sheet (2007) http://www.seagate.com/docs/pdf/datasheet/ disc/ds_barracuda_es.pdf.



(a) mean read request, $\lambda = 0.01$



(b) mean write request, $\lambda = 0.02$

Fig. 10. Plot of mean response time against request size on a 4 disk RAID 5 system (λ requests/ms)